Bulletin of Electrical Engineering and Informatics
Vol. 10, No. 6, December 2021, pp. 3256~3264
ISSN: 2302-9285, DOI: 10.11591/eei.v10i6.2865

Transforming data-centric eXtensible markup language into relational databases using hybrid approach

Su-Cheng Haw, Emyliana Song
Faculty of Computing and Informatics, Multimedia University, 63100 Cyberjaya, Malaysia

Article Info

Article history:

Received Apr 17, 2021
Revised Jul 19, 2021
Accepted Oct 12, 2021

Keywords:

Model-based mapping
XML database
XML labelling
XML to RDB
XML transformation

ABSTRACT

eXtensible markup language (XML) appeared internationally as the format for data representation over the web. Yet, most organizations are still utilising relational databases as their database solutions. As such, it is crucial to provide seamless integration via effective transformation between these database infrastructures. In this paper, we propose XML-REG to bridge these two technologies based on node-based and path-based approaches. The node-based approach is good to annotate each positional node uniquely, while the path-based approach provides summarised path information to join the nodes. On top of that, a new range labelling is also proposed to annotate nodes uniquely by ensuring the structural relationships are maintained between nodes. If a new node is to be added to the document, re-labelling is not required as the new label will be assigned to the node via the new proposed labelling scheme. Experimental evaluations indicated that the performance of XML-REG exceeded XMap, XRecursive, XAncestor and Mini-XML concerning storing time, query retrieval time and scalability. This research produces a core framework for XML to relational databases (RDB) mapping, which could be adopted in various industries.

Corresponding Author:

Su-Cheng Haw
Faculty of Computing and Informatics
Multimedia University
Jalan Multimedia, 63100 Cyberjaya, Malaysia
Email: sucheng@mmu.edu.my

1. INTRODUCTION

In today's information age, technology is taking precedence over traditional way of getting work done. Technology is advancing every minute of the day and millions of data are being produced each second. For instance, the social media such as Twitter and Facebook tags data using XML. Information in XML format is then exported or imported to make it usable and standardized for others to use [1]-[3]. Hence, a significant amount of data is generated and need to be properly processed by organizations for data storage and manipulation. XML is often used to distribute data over the internet because this format eases data exchange process. Due to the advantages of XML, many organizations have choosen XML as the standard format for business transaction [4].

On database management trend, the ability on processing various types of data such as structured, semi-structured, and unstructured data has become important [5]. Although native XML databases do present, the migration cost from the previous database storage management into XML-based is not a come-and-go category. There are various possible underlying storages such as big data, temporal database, object-oriented database, relational database (RDB) and object-relational database [6]. Nevertheless, the focus of this paper is only on RDB as it is the most widely used back-end database in many organizations.

Subsequently, many attempts to come out with the efficient mapping scheme between the two technologies has emerged [7].

The mapping scheme of XML to RDB are categorized into four main groups, namely, edge-based, node-based, path-based and hybrid-based scheme [8]. Among these, the edge-based scheme is the easiest, whereby all the edges are stored in a table. Nevertheless, this technique requires huge storage space and may requires several self-join within the table to answer complex queries. The path-based scheme tracks the hierarchical path information [9]. For the storage, it consists of two tables, whereby the first table stores the path information on non-leaf nodes, while the second table properties the leaf nodes path information. As such, depending on the query expressed, it could involve retrieving the results from either table or both tables. On the contray, the node-based scheme annotate each node to denote the absolute position of the node in the XML tree. To be able to identify the nodes and their associated relationship uniquely, this technique may assign label that may become overhead as the size of the XML tree grows. The hybrid-based scheme, however, is the combinative of several techniques. Bousalem and Cherti [10] proposed XMap to transform XML into RDB based on ORDPATH scheme [11] to annotate the data in XML document. The authors done the theoretical comparisons between Edge [12], XRel [13], XParent [14] and XMap. Theoretically, it shows their approach leads in support for dynamic updates as compared to other approaches. Fakharaldien *et al*. [15] proposed XRecursive as the mapping scheme between XML and RDB. XRecursive used the parent id information to identify the path to each node recursively. Experimental evaluation indicated that XRecursive performed better as compared SUXCENT [16] as it only uses two tables while SUCCENT uses five tables.

Qtaish and Ahmad [8] proposed XAncestor, which has the uniqueness of storing the path information in a pre-defined scheme. This reduces the necessary storage size. It is most significant when it involves a huge size of the dataset. The experimental evaluation demonstrated that XAncestor has the most robust storage time and space as compared to XRel [13], XRecursive [15], s-XML [17], SMX/R [18] and Ying *et al*. approach [19]. Zhu *et al.* proposed a path-based mapping scheme, Mini-XML [20], which stores leaf nodes independently from the data table. They adopted the persistent labelling scheme [21] to annotate each node as the traversal proceeds in depth-first manners. Experimental evaluations were compared against s-XML [17] on various dataset sizes (range of 2.2 MB to 683 MB). The results exhibited that Mini-XML achieved better performance in storage time and storage space. In a more recent study, Hsu and Liao [22] proposed a compact indexing scheme named UCIS-X based on a branch map. The branch map preserves the mapped information between parent and child nodes without the need to annotate each node. In another separate research, Taktek and Thakker proposed a pentagonal scheme based on prefix labelling to capture the structural relationships. As a result, it is not needed to access the physical document during query processing [23]. The summary of some approaches is depicted in Table 1.

Table 1. Comparison on various approaches

Approach	Number of Table	Advantages	Disadvantages
XMap [10]	3	Utilise the ORDPath scheme [9] that support for dynamic support.	Redundant attributes stored in data table.
XRecursive [15]	2	Good for query retrieval involving P-C relationship.	Suffers from query retrieval that involved recursive search and A-D relatinoship.
XAncestor [8]	2	Utilise the pre-define scheme for more efficient storage and retrieval.	Labelling technique does not support certain types of queries.
s-XML [17]	2	Efficient query retrieval over large datasets.	Redundant attributes stored in tables.
Ying *et al.* [19]	4	Utilise the path table for efficient storage and retrieval.	Storing unnecessary leaf node path information.
Mini-XML [20]	2	Utilise the persistent labelling scheme for dynamic updates.	Redundant attributes stored in tables.

In this section, we have reviewed the XML mapping schemes and the labelling technologies. It shows that the recent few approaches are adopting path-based approach and some even proposed hybrid system, which comprises of node-based and path-based schemes. From the review thus far, we noticed that the number of tables used, affect highly on query retrieval time. This is due to the join operation required to acquire desired query results. A rule of thumb for our proposed mapping scheme is to have the minimum number of tables and yet rich enough to provide necessary information to support assorted query retrieval. As such, we propose a new mapping scheme named XML-REG, where the REG represents Region, which is the labelling scheme under the region-based grouping. The following section consists of further elaboration on our proposed scheme.

2. RESEARCH METHOD

2.1. The system architecture

Figure 1 illustrates overview of system architecture design. The processes involved are reading the XML document, XML mapping (tree annotation and data transformation) and query retrieval. Firstly, XML document will be read by an event-driven parser, StAX parser [24]. Being an event-based triggers parser, StAX does not have to wait for the document to be full load in order to parse the document. This is also affected for the fact that it is a Pull API where it pulls data that client request. None of the data will be stored into the memory. Thus, for data to be re-read, one must parse the document all over again. StAX reads each tag and data, then create an event that calling program can use.

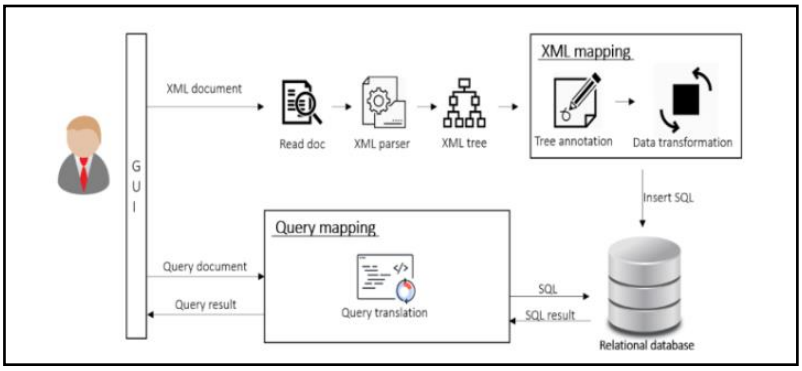


Figure 1. Overview of system architecture design

According to Khanjari and Gaeini [25], a labeling scheme is efficient if "it is small size labels keeping the simplicity of the exploited algorithm in order to avoid complex computations as well as retaining the readability of structural relationships between nodes". We designed the labeling scheme based on this definition. Figure 2 shows the XML data model annotated with XML-REG. Each node will be traversed in depth-first order to annotate the label based on its position. Each node is represented as (l, s, e) label, whereby l is the level of the node, s is the startid, and e is the endid. The nodes are said to be in Parent-Child (P-C) relationship, if the difference within the level of parent and child node is one. Conversely, the nodes are said to be in ancestor-descendant (A-D) under two conditions: (i) level difference of more than one, and (ii) the nodes must be in a range (region). For example, to identify if node 5 has A-D relationship with node 1, the id of node 5 must be within the range of startid and endid of node 1.

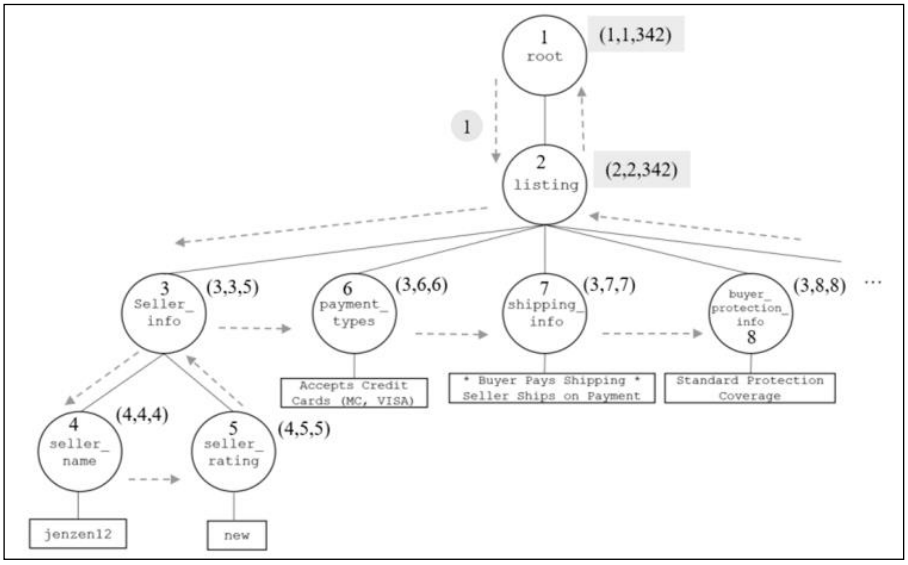


Figure 2. XML document labelled with XML-REG labeling scheme

Dynamic updates can be identified in a few groups, such are insertion of new node, node deletion and edit on node information. Nevertheless, the edit and delete operations are pretty clear-cut as there will not be any changes on the node positioning. Insertion on the other hand, is the most crucial operation as it may causes the whole XML to be regenerated due to changes on the node positioning. Henceforth, the focus on this paper is on insertion operation on the dynamic updates. Figure 3 exemplifies the three possible insertion situations: (i) leftmost, (ii) rightmost and (iii) in-between. For any leftmost insertion, the value of startid will be appended with '.0' to the end of the initial label. In this example, since the leftmost node is with label 2, the node to be inserted will be labeled as 2.0 (see node A). For subsequent leftmost insertion, for instance, if node B is to be inserted, it will be added '.0'. Thus, the label for node B is 2.00. In the rightmost insertion case, the id given label will be added into the value table as it is. For instance, Node D will be labeled with 6. On the other hand, for an in-between insertion, in this case, node F, the node will be added .0 at the end of the node label.

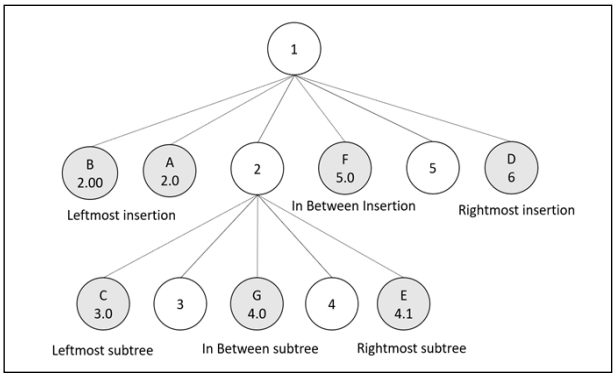


Figure 3. Dynamic updates based on XML-REG labeling scheme

In XML-REG, two tables, namely, (i) Element_path table, and (ii) Value table are created. By having the minimum number of tables, and yet sufficient to facilitate the join operation, it could save the storage space while enable complex query to be supported. The Element_ path table stores all distinct path information of nodes in the XML document. Figure 4 (a) depicts the snippet of the Element_path table, while Figure 4 (b) depicts the snippet of the Value table. The ancestor and parent nodes are identified based on the RPathId and PathId attributes in the Value and Element_path tables respectively.

PathId	Pathexp
1	/root
2	/root/listing
3	/root/listing/seller_info
4	/root/listing/seller_info/seller_name
5	/root/listing/seller_info/seller_rating
6	/root/listing/shipping_info
7	/root/listing/payment_types

(a)

Level	ID	Value	RPath Id
4	4	jenzen12	4
4	5	new	5
3	6	* Buyer Pays Shipping * Seller Ships on Payment	6
3	7	Accepts Credit Cards (MC, VISA)	7

(b)

Figure 4. Snippet of, (a) element_path, (b) value table

2.2. XML-REG implementation

Figure 5 shows the algorithm of XML-REG annotation. The algorithm takes in an XML file and output the annotated XML file based on XML-REG labeling scheme (line 2 to 3). Firstly, the connection to the database needs to be established as in line 4. Next, the chosen XML dataset is to be loaded as in line 5. In the algorithm, a stack named stackPath is constructed to keep path information in a hierarchical manner commencing from root to the current node (see line 5). Subsequently, the StAX parser is activated through the function getEventType to return the type of the node (see line 9, line 30 and line 37).

The startElement retrieves the elements and attributes that exist within the angle bracket tag (< >). If the elements are encountered, it retrieves the respective element name and id to stored it in variable qName. The element name will then be concatenate to form the path. Yet, if the path is found in the path table, it will

not be stored in stackPath. On the other hand, if the attributes are encountered, the information will be stored into Value table with RPathid formed in the stackPath. The other EventType is character. When this tag is encountered, the text node information will be saved into Value table. As for the endElement, which is encountered when the </ > is reach, the end qName in string path will be removed and level will be subtracted by 1.

2.3. Experimental evaluation

Four existing approaches will be implemented, namely Mini-XML [20], XAncestor [8], XMap [10] and XRecursive [15]. Unlike XML-REG, XRecursive uses node-based mapping technique, while XAncestor and Mini-XML approach uses the path-based mapping technique to store XML data into RDB.

All experiments were evaluated on AMD Ryzen 7 processor with the RAM of 32 GB and memory of 237 GB. The system is implemented in Java SE Development Kit, while the RDBMS is in Microsoft SQL Server. The experiments are carried out on DBLP original dataset (130.73 MB) [26] on the query evaluation, while to prove the scaliability of our proposed approach, we demonstrate the behavior of XML-REG across DBLP original dataset, while increasing the scale up to 15 times, DBLP15 (1.99 GB) as depicted in Table 2. The DBLP dataset is selected based on two main reaons: (1) the depth of the dataset should be at least three level in order to test for the A-D relationship, and (2) the dataset should contains attributes in order to demonstrate the query retrieval with attribute as the constraints. In terms of query, there are two classes of queries, namely path query (simple query) and twig query (complex query with branching nodes) [27].

```
Algorithm 1: XML-REG annotation
1.  Function createXML-REG {
2.  Input: An XML document
3.  Output: Annotated XML document
4.  Establish database connection
5.  Stack stackPath = new Stack()
6.  Create table
7.  Get event type
8.
9.  If is start element
10.    Id++
11.    Level++
12.    qName = startElement.getName()
13.    Path = currentpath + "/" + qName
14.    If lstackPath.contain(path)
15.        Pathid++
16.        stackPah.add(path(
17.             While (attributes.hasNext())
18.            Id++
19.            Level++
20.            attName = "@" + attr.getName()
21.             strPath = path + "/" + attribute
22.            If !stackPath.contains(path)
23.                Path++
24.                stackPath.add(path)
25.                Pathvalue = stackPath.IndexOf(attrPath) + 1
26.       Insert into value table
27.       Level = level - 1
28. End If
29.
30. If character
31.      elementValue = characters.getData();
32.     If !elementValue.isempty[]
33.        Pathvalue = stackPath.IndexOf(path)  + 1
34.      Insert into Value table
35. End If
36.
37. If end element
38.     Pathlist = path.substring(path.lastIndexOf("/") + ;
39.   If pathlast.equals(eName)
40.       Path = path.substring(0, path. lastIndexof("/"))
41.       Level = level - 1
42. End If
43. InsertPath()
44. } //End function
```

Figure 5. XML-REG pseudocode

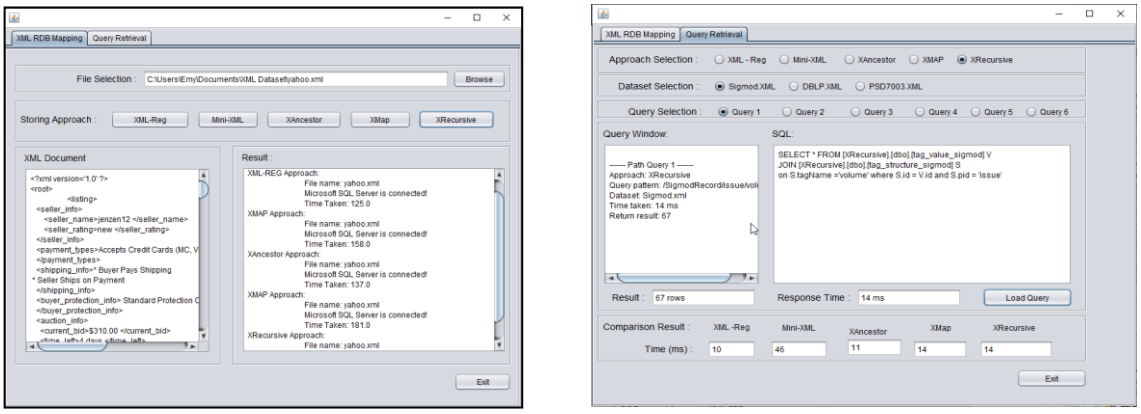
Table 2 DBLP dataset on various sizes

Dataset	DBLP	DBLP5	DBLP10	DBLP15
Data size	130.726 MB	653.625 MB	1307.25 MB	1960.874 MB

In the first section of evaluation, data storing, the XML document is mapped and transformed into RDB storage. The structure and number of tables is designed uniquely in each approach with the aim to have an efficient and yet lossless transformation. Thus, the time taken for each approach to map and to complete will be recorded. In addition, the storage size after each mapping approach will also be recorded. To achieve higher accuracy of time taken to store, each mapping approach will be executed six times, and the average time is taken as the result the five-consecutive run, excluding the first run as the first run usually involved some buffer time.

For the second part of the evaluation test, each approach will be put to test on the duration it takes to retrieve various queries from the stored RDB after each mapping approach. The results were varying due to the amount of join operation uses by each approach considering the design of its table and how data were stored in the tables. Two types of query design were prepared to assess these approaches that will cover P-C, A-D and mixed relationships. These queries consist of three paths and three twig queries. Last but not least, our proposed approach, XML-REG was put on some tests in order to demonstrate that XML-REG is able to support dynamic updates. These tests include right-most insertion, left-most insertion, and in-between insertion.

Figure 6 (a) and Figure 6 (b) exhibit the simulation engine for performance evaluation on data storing as well as query retrieval process. Figure 6 (a) shows XML-RDB mapping tab, which is used for data storing evaluation. The user clicks on the browse button to choose a dataset to be transformed into RDB storage. On the left text area, the selected XML document content will be illustrated. The result of the evaluation will be presented in the result text area (on the right text area). Figure 6 (b) depicts the query retrieval tab in the simulation engine. Firstly, the user will select the mapping approach, followed by dataset and query selection respectively. Once the query is selected, the corresponding SQL statement will be displayed in the SQL window. User can then click the "Load Query" button to confirm their selection. As such, the time taken for the query execution and the number of return results will be presented at the interface. In addition, the bottom part of the interface also outlines the time taken by other approaches to perform the same evaluation.



(a) (b)

Figure 6. These figures are; (a) data storing process, (b) query retrieval process on simulation engine

3. RESULTS AND DISCUSSIONS

Three evaluation involving data storing time, query retrieval, and scalability were recorded. The result is discussed in details in sub-section according to the evaluation type.

3.1. Results on data storing evaluation

The experiment is repeated seven times with the first reading being eliminated to avoid inaccuracy due to buffering effects. Subsequently, the final result is generated as the average of six times consecutive runs. Table 3 depicts the data strong results. We observed that XML-REG has the best storing time, followed by XMap [10], Xancestor [8], Mini-XML [20] and XRecursive [15]. As the dataset increases, the

competence of the approach can be observed obviously. XRecursive suffers greatly as it recursively calls the child node and stores irrelevant data into the database. In contrast, XML-REG stores minimum information, that is the unique path of the respective node and the values based on the proposed unique label id.

Table 3. Data storing evaluation on various approaches

Insertion time (minute)				
XML-Reg	Mini-XML	XAncestor	XMap	XRecursive
11.23	18.58	15.72	15.01	26.16

3.2. Results on query retrieval evaluation

Table 4 shows the query response time for the DBLP dataset. The evaluation outcomes indicated that XML-REG is the best, succeeded by XMap, XAncestor, Mini-XML and XRecursive. It is noteworthy that XRecursive unable to support any query that involved the A-D relationship. This is because, in the XRecursive approach, one will need to recursively find the ancestor and parent node. This is impossible as one does not know the nodes that existed in between the ancestor to the particular node. As such for query with the A-D relationship (PQ2, PQ3, TQ2 and TQ3), the remark 'not supported' is placed in the respective column. It can be observed that XMap is faster than XAncestor because it requires fewer joins. Nevertheless, for query with high complexity such as TQ3, XML-REG performs the best followed by XAncestor, XMap and finally Mini-XML. XMap employed three tables in the storage; as such, the number of join operation are far more as it requires to firstly, find the join path among path and vertex tables, and consequently, find the intersection join to retrieve the query match.

Table 4 Query retrieval evaluation on various approaches

	Approaches (ms)				
	XML-REG	Mini-XML	XAncestor	XMap	XRecursive
PQ1	433	1047.8	882.6	1237.4	1610.2
PQ2	402.4	5530	768.4	1294	not supported
PQ3	400	1049.8	884.2	458.4	not supported
TQ1	1246.4	1826	1661.6	3979.8	8715
TQ2	1791.6	5699.4	4350.8	4168.8	not supported
TQ3	1229.6	10764.2	1899.6	4198.2	not supported

3.2. Results on scaling evaluation

The scalability test indicates the efficiency of how each approach handles large-scale datasets. Concerning this, the DBLP dataset is multiplied by 5 times for each iteration (DBLP1, DBLP5, DBLP10 and DBLP15) to show the scalability of each approach. Figure 7 expresses the time taken for the storage evaluation on various sizes of DBLP. It is observed that XML-REG shows the best performance as the line graph is almost flat with the increment of DBLP sizes. XRecursive approach, however, has the sharpest increase with the growth of the DBLP sizes. As such, it is the least scalable.

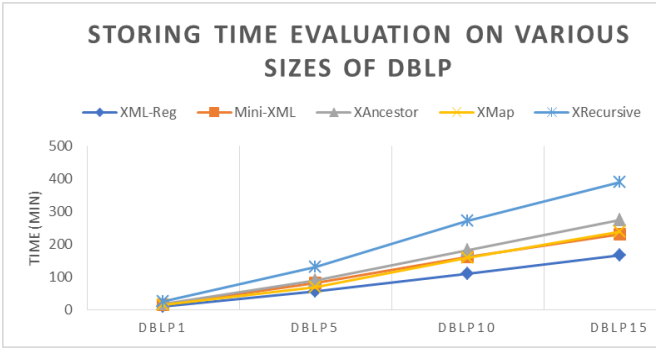


Figure 7. Result of path query on various DBLP datasets

4. CONCLUSION

In this paper, we proposed XML-REG, which utilized the hybridisation of the best features of path-based and node-based approaches to map XML to RDB storage. Experimental evaluations demonstrated that XML-REG exhibits the most excellent outcomes for all the evaluations: (i) data storing time and (ii) query retrieval time as compared to XMap, Xancestor, Mini-XML, and XRecursive. In addition, in the scalability test, the complexity of XML-REG is O(n), indicating that XML-REG is scalable to support huge datasets. The results also indicated the workable hybrid approach for effective mapping. In our future work, we propose to look into the possible way of XML compression technology to reduce the label size to accommodate for huge insertion. To improve the performance, we will implement XML-REG in the distributed environment.

REFERENCES

[1] P. Singh and S. Sachdevaa, "A Landscape of XML Data from Analytics Perspective," *Procedia Computer Science*, vol. 173, pp. 392-402, 2020, doi: 10.1016/j.procs.2020.06.046.

[2] X. Lin *et al.*, "A Fast Filtering Method of Invalid Information in XML File," *Big Data Analytics for Cyber-Physical System in Smart City*, 2020, pp 259-264, doi: 10.1007/978-981-33-4572-0_38.

[3] F. Azzedin, S. Mohammed, M. Ghaleb, J. Yazdani and A. Ahmed, "Systematic Partitioning and Labeling XML Subtrees for Efficient Processing of XML Queries in IoT Environments," in *IEEE Access*, vol. 8, pp. 61817-61833, 2020, doi: 10.1109/ACCESS.2020.2984600.

[4] K. T. Chau, Q. He, X. Hu and R. Wu, "Comparison on Performance of Text-Based and Model-Based Architecture in Open Source Native XML Database," *2019 IEEE 4th International Conference on Signal and Image Processing (ICSIP)*, 2019, pp. 340-344, doi: 10.1109/SIPROCESS.2019.8868709.

[5] A. H. Al-Hamami and A. A. Flayyih, "Enhancing Big Data Analysis by using Map-reduce Technique," *Bulletin of Electrical Engineering and Informatics*, vol. 7, no. 1, pp. 113-116, 2018, doi: 10.11591/eei.v7i1.895.

[6] F. Arena and G. Pau, "An overview of big data analysis," *Bulletin of Electrical Engineering and Informatics*, vol. 9, no. 4, pp. 1646-1652, 2020, doi: 10.11591/eei.v9i4.2359.

[7] H. Nassiri *et al.*, "Integrating XML to Relational Data," *Procedia Computer Science*, vol. 110, pp. 422-427, 2017, doi: 10.1016/j.procs.2017.06.107

[8] A. Qtaish and K. Ahmad, "XAncestor: An efficient mapping approach for storing and querying XML documents in relational database using path-based technique," *Knowledge-Based Systems*, vol. 114, pp. 167-192, 2016, doi: 10.1016/j.knosys.2016.10.009.

[9] S. Jawari Kapisha and G. Vijaya Lakshmi, "Exploring XML Index Structures and Evaluating C-Tree Index-based Algorithm," *2020 3rd International Conference on Intelligent Sustainable Systems (ICISS)*, 2020, pp. 212-218, doi: 10.1109/ICISS49785.2020.9316052.

[10] Z. Bousalem and I. Cherti, "XMap: A Novel Approach to Store and Retrieve XML Document in Relational Databases," *Journal Of Software*, vol. 10, no. 12, pp. 1389-1401, 2015, doi: 10.17706/jsw.10.12.1389-1401.

[11] P. O'Neil *et al.*, "ORDPATHs: insert-friendly XML node labels," *ACM SIGMOD International Conference on Management of Data*, 2004, pp. 903-908, doi: 10.1145/1007568.1007686.

[12] D. Florescu and D. Kossmann, "Storing and Querying XML Data using an RDMBS," *IEEE Data Engineering Bulletin*, vol. 1060, no. 22, pp. 27-34, 1999.

[13] M. Yoshikawa *et al.*, "XREL: A Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases," *ACM Transactions on Internet Technology*, vol. 1, no. 1, pp. 110-141, 2001, doi: 10.1145/383034.383038.

[14] H. Jiang, H. Lu, W. Wang and J. X. Yu, "XParent: an efficient RDBMS-Based XML database system," *Proceedings 18th International Conference on Data Engineering*, 2002, pp. 335-336, doi: 10.1109/ICDE.2002.994745.

[15] M. A. I. Fakharaldien, J. M. Zain and N. Sulaiman, "XRecursive: AStorage Method for XML Document Based on Relational Database," in *International Conference on Software Engineering and Computer Systems*. Springer, Berlin, Heidelberg, 2011, doi: 10.1007/978-3-642-22191-0_40.

[16] P. Sandeep, "Efficient storage and query processing of XML data in relational database systems," Master's thesis, Nanyang Technological University, Singapore, 2005.

[17] S. Subramaniam *et al.*, "s-XML: An efficient mapping scheme to bridge XML and relational database," *Knowledge-Based Systems*, vol. 27, pp. 369-380, 2012, doi: 10.1016/j.knosys.2011.11.007.

[18] F. Abduljwad, W. Ning and X. De, "SMX/R: Efficient way of storing and managing XML documents using RDBMSs based on paths," *2010 2nd International Conference on Computer Engineering and Technology*, 2010, pp. V1-143-V1-147, doi: 10.1109/ICCET.2010.5486247.

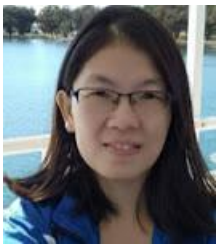
[19] J. Ying, S. Cao and Y. Long, "An efficient mapping approach to store and query XML documents in relational database," *Proceedings of 2012 2nd International Conference on Computer Science and Network Technology*, 2012, pp. 2140-2144, doi: 10.1109/ICCSNT.2012.6526341.

[20] H. Zhu, H. Yu, G. Fan and H. Sun, "Mini-XML: An efficient mapping approach between XML and relational database," *2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS)*, 2017, pp. 839-843, doi: 10.1109/ICIS.2017.7960109.

[21] A. Gabillon and M. Fansi, "A new Persistent Labelling Scheme for XML," *Journal of Digital Information Management*, vol. 4, no. 2, pp. 112-116, 2006.

[22] W. -C. Hsu and I. -E. Liao, "UCIS-X: An Updatable Compact Indexing Scheme for Efficient Extensible Markup Language Document Updating and Query Evaluation," in *IEEE Access*, vol. 8, pp. 176375-176392, 2020, doi: 10.1109/ACCESS.2020.3025566.
[23] E. Taktek and D. Thakker, "Pentagonal scheme for dynamic XML prefix labelling," *Knowledge-Based Systems*, vol. 209, p. 106446, 2020, doi: 10.1016/j.knosys.2020.106446.
[24] Sun Microsystems, "Streaming APIs for XML Parsers, Java Web Services Performance," Team White Paper, retrived on Jan 2021.
[25] E. Khanjari and L. Gaeini, "A new effective method for labeling dynamic XML data," *Journal of Big Data*, vol. 5, no. 50, pp. 1-17, 2018, doi: 10.1186/s40537-018-0161-4.
[26] UW. Washington University XML repository, retrived on Jan 2021. http://aiweb.cs.washington.edu/research/projects/xmltk/xmldata/www/repository.html.
[27] S. Subramaniam, S. -C. Haw and L. -K. Soon, "Improved Centralized XML Query Processing Using Distributed Query Workload," in *IEEE Access*, vol. 9, pp. 29127-29142, 2021, doi: 10.1109/ACCESS.2021.3058383.

BIOGRAPHIES OF AUTHORS



Su-Cheng Haw is Associate Professor at Faculty of Computing and Informatics, Multimedia University, where she leads several funded researches on the XML databases. Her research interests include XML databases, query optimization, data modeling, semantic web, and recommender system.

Emyliana Song is a postgraduate student at Faculty of Computing and Informatics in Multimedia University. She is currently doing a research on XML data mapping scheme in transforming XML document into relational database.